ZARZĄDZANIE PAMIĘCIĄ

Standardowe funkcje **malloc** i **calloc** dynamicznie pobierają od systemu żądane bloki pamięci.

```
void *malloc(size_t n);
```

Zwraca:

- wskaźnik do n bajtow nie zainicjowanej pamięci w przypadku powodzenia
- NULL, gdy żądanie nie może być spełnione.

Przykład:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <process.h>

int main(void)
{
  char *str;
  /* allocate memory for string */
  if ((str = (char *) malloc(10)) == NULL)
  {
    printf("Not enough memory to allocate buffer\n");
    exit(1);  /* terminate program if out of memory */
  }
  /* copy "Hello" into string */
  strcpy(str, "Hello");
  /* display string */
  printf("String is %s\n", str);
  /* free memory */
  free(str);
  return 0;
}
```

```
void *calloc(size_t obj, size_t size);
```

Funkcja **calloc** zwraca wskaźnik do obszaru pamięci przeznaczonego dla tablicy złożonej z *nobj* elementów, każdy o rozmiarze *size*.

Funkcja zwraca **NULL**, jeśli to polecenie nie może być wykonane. Obszar jest inicjowany zerami.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
   char *str = NULL;
   /* allocate memory for string */
   str = (char *) calloc(10, sizeof(char));
   /* copy "Hello" into string */
   strcpy(str, "Hello");
   /* display string */
   printf("String is %s\n", str);
   /* free memory */
   free(str);
   return 0;
}
```

Uwaga: Wartość wskaźnika zwracanego przez obydwie funkcje trzeba zrzutować na odpowiedni typ.

Zmiana wielkości przydzielonego obszaru

```
void *realloc(void *p, size_t size);
```

Funkcja *realloc* zmienia rozmiar obiektu wskazywanego przez **p** na wartość określoną przez **size**. Zawartość obiektu nie ulegnie zmianie w jego części początkowej o rozmiarze równym mniejszemu z rozmiarów: starego i nowego. Jeśli nowy rozmiar jest większy, to dodatkowy obszar pamięci nie jest inicjowany.

Funkcja zwraca wskaźnik do nowego obszaru lub NULL, jeśli polecenie nie może być wykonane. Wówczas wskaźnik **p** nie ulega zmianie.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
  char *str;
  /* allocate memory for string */
  str = (char *) malloc(10);
  /* copy "Hello" into string */
  strcpy(str, "Hello");
  printf("String is %s\n  Address is %p\n", str, str);
  str = (char *) realloc(str, 20);
  printf("String is %s\n  New address is %p\n",
          str, str);
  /* free memory */
  free(str);
  return 0;
}
```

Zwolnienie przydzielonej pamięci

```
void free(void *p);
```

Funkcja *free* zwalnia obszar pamięci wskazywany przez *p*; nie robi nic, jeśli *p* równa się NULL.

Argument *p* musi być wskaźnikiem do obszaru uprzednio przydzielonego przez jedną z funkcji: *malloc, calloc, realloc.*

Różnica między standardową deklaracją tablicy, a jej dynamiczną alokacją

```
1)    char hello[5];

2)    char *hello=(char *) malloc((size_t) 5);
```

1. pamięć przydzielana do pamięci stosu (stack memory), która jest ponownie wykorzystywana po zakończeniu działania funkcji
2. przydział do pamięci sterty - heap (heap memory - sterta zmiennych dynamicznych w pamięci operacyjnej.

Heap memory - cała pamięć za wyjątkiem bufora pamięci stosu (oraz bufora bezpieczeństwa wokół stosu.

Uwaga:

- błędem jest zwalnianie czegos, co nie zostało przydzielone za pomoą funkcji *malloc* lu/b *calloc*
- błędem jest również używanie czegoś, co zostało zwolnione

Typowym i niepoprawnym fragmentem programu jest następująca pętla, która zwalnia bloki pamięci powiązane w łańcuch:

```
for (p=head; p != NULL; p=p->next)  /* Zle  */
  free(p);

Poprawnie nalezy przechowywac wszystko, co jeszcze
bedzie potrzebne przed zwolnieniem pamieci:

for (p=head; p != NULL; p=q) {
  q=p->next;
  free(p);
}
```

Tablice wielowymiarowe

Zasady:

- tablica dwuwymiarowa jest jednowymiarową tablicą, w której każdy element jest tablicą jednowymiarową
- dopuszczalne jest niedookreślenie pierwszego wymiaru (jego określenie następuje wówczas w momencie inicjalizacji), natomiast następne muszą być sprecyzowane
- elementy są umieszczane w pamięci wierszami, tzn. odwołanie

```
daytab[i][j] /* [wiersz][kolumna] */
oznacza odniesienie do wiersza i-tego,
kolumny  j-tej. (Najszybciej zmienia
si\c e prawy skrajny indeks.
```

Przykład:

```
static char daytab[2][13]={
 {0,31,28,31,30,31,30,31,31,30,31,30,31},
 {0,31,28,31,30,31,30,31,31,30,31,30,31}};

/* podaj dzien roku na podstawie miesiaca i dnia  */
int day_of_year(int year, int month, int day)
{
  int i,leap;
  leap=year % 4 == 0 && year % 100 != 0
                     || year % 400 == 0;
  for (i=1; i<month; i++)
    day += daytab[leap];
  return day;
}

/* podaj miesiac i dzien na podstawie dnia roku */
void month_day(int year, int yearday,
               int *pmonth, int *pday)
{
  int i,leap;
  leap=year % 4 == 0 && year % 100 != 0
                     || year % 400 == 0;
  for (i=1; yearday > daytab[leap][i]; i++)
    yearday -= daytab[leap][i];
  *pmonth=i;
  *pday=yearday;
}
```

Inicjowanie tablic wielowymiarowych

Lista inicjatorów jej elementów ujęta w nawiasy klamrowe.

- Jeśli w pewnym nawiasie klamrowym zabraknie elementów, to uzupełnia się zerami (`{0}`).
- Jeśli w pewnych nawiasach wewnętrznych zostaną wymienione wszystkie inicjatory, to nawiasy takie można pominąć.
- nawiasy klamrowe zawierające listę zmaków można zastąpić napisem-łańcuchem składającym się z takich właśnie znaków.
- opuszczenie w deklaracji tablicy określenia liczby elementów w pierwszym wymiarze tablicy powoduje domniemanie go na podstawie listy inicjatorów.

Przykłady:

```
int Vec[3]={10,20,30};
long int Arr[3][2]={ {1,2}, {3,4}, {5,6}};
char Greet[6]={'H','e','l','l','o'};
/*  Greet[6] przypisano domy\' slnie '\0' */
char Text[]="Hello World";
/*  domniemany rozmiar  12 */
short int Matrix[2][3]={{1,2},{3}};
/*  domniemany inicjator {{1,2,0},{3,0,0}} */
```

Wskaźniki a tablice wielowymiarowe

Po następujących definicjach:

```
int a[10][20];
int *b[10];
```

oba zapisy *a[3][4]* i *b[3][4]* są poprawnymi odwołaniami do pojedyńczego obiektu typu **int**.

a jest prawdziwą tablicą wielowymiarową, zarezerwowano dla niej 200 miejsc o rozmiarze **int**; element *a[wiersz][kolumna]* znajduje się według wzoru: *20*wiersz+kolumna*

b przydziela jedynie 10 miejsc na wskaźniki i nie inicjuje ich; nadanie wartości początkowych musi być zrobione jawnie - statycznie lub programowo.

Jeśli każdy z elementów tablicy *b* wskazuje na tablicę 20 elementów całkowitych. Wówczas mamy zarezerwowane 200 miejsc rozmiaru **int** plus 10 komórek na wskaźniki.

Ważną przewagą tablicy wskaźników jest możliwość **zróżnicowania długości wierszy**.

Przekazywanie tablicy dwuwymiarowej do funkcji

Musimy podać liczbę kolumn. Zatem, jeśli tablica *daytab* ma być przekazana do funkcji *f*, to deklaracja funckji powinna mieć jedną z trzech poniższych postaci:

```
f(int daytab[2][13]);

f(int daytab[][13]);

f(int (*daytab)[13]);
```

Ostatnia deklaracja mówi, że *daytab* jest wskaźnikiem do tablicy 13 liczb całkowitych. Nawiasy okrągłe są w tym przypadku konieczne, ponieważ nawiasy kwadratowe [] mają wyższy priorytet niż operator adresowania pośredniego *. Bez nawiasów okrągłych deklaracja

```
int *daytab[13];
```

wprowadza tablicę 13 wskaźników do obiektów całkowitych.

Możliwość korzystania z tablic wielowymiarowych o indeksach ujemnych

Rozwiązanie przyjęte przez autorów **Numerical Recipes**:

```
#include <stdlib.h>
#include <stdio.h>
#define TYPFLOAT long double
#define MSQRT sqrtl
#define MFABS fabsl
```

```
void nrerror(char error_text[])
{
 fprintf(stderr,
        "Numerical Recipes run-time error...\n");
 fprintf(stderr,"%s\n",error_text);
 fprintf(stderr,"...now exiting to system...\n");
 exit(1);
}

TYPFLOAT **dmatrix(int nrl,int nrh,int ncl,int nch)
{
 int i;
 TYPFLOAT **m;

 m=(TYPFLOAT **)
   malloc((unsigned) (nrh-nrl+1)*sizeof(TYPFLOAT*));
 if (!m) nrerror("allocation failure 1 in dmatrix()");
 m -= nrl;

 for(i=nrl;i<=nrh;i++) {
   m[i]=(TYPFLOAT *)
      malloc((unsigned) (nch-ncl+1)*sizeof(TYPFLOAT));
   if (!m[i])
     nrerror("allocation failure 2 in dmatrix()");
   m[i] -= ncl;
 }
 return m;
}

void free_dmatrix(TYPFLOAT **m,
   int nrl,int nrh,int ncl,int nch)
{
 int i;

 for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
 free((char*) (m+nrl));
}
```

Odwołania do funkcji *dmatrix* mogą wyglądać następująco:

```
TYPFLOAT **Q;

/* .......  */
Q=dmatrix(-10,N,-5,N);
```

Struktury odwołujące się do samych siebie - tworzenie listy

Lista - uporządkowany ciąg elementów, z których każdy zawiera wskaźnik do następnego elementu.
Drzewo - uporządkowany ciąg elementów, z których każdy zawiera wskaźnik do następnego i poprzedzającego elementu.

Przykład (program tworzący listę danych wczytywanych z klawiatury i wypisujący ją):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define DL_NAZW 20  /* maks. dl. nazwiska */

typedef struct element /* def. typu element */
   {  char nazw[DL_NAZW+1]; /*nazwisko */
      int wiek;             /* wiek */
      struct element *nast; /* wsk. do nast. el. */
   } t_element;

void main()
{  void czytanie(t_element **); /* f. tworzaca */
   void pisanie(t_element *);   /* f. wypisujaca*/
   t_element *poczatek; /*wsk. do poczatku listy*/
   czytanie(&poczatek);
   pisanie(poczatek);
}

/* Funkcja odczytu i tworzenia listy */
void czytanie(t_element **adpocz)
{  char nazwwe[DL_NAZW+1]; /* nazwisko czytane */
   t_element *temp;        /* na zamiane wsk. */
   *adpocz=NULL;    /* na poczatku lista pusta */
   while (1) {
      printf("nazwisko: ");
      gets(nazwwe);
      if (strlen(nazwwe)) {
         temp=(t_element *) malloc(sizeof(t_element));
         strcpy(temp->nazw,nazwwe);
         printf("wiek: ");
         scanf("%d", &temp->wiek);
         getchar(); /* przeskakuje znak \n */
         temp->nast=*adpocz;
         *adpocz=temp;
      }
      else break; /* wyjscie, gdy nazwisko puste */
   }
}
/* Funkcja wyprowadzania zawartosci listy */
void pisanie(t_element *poc)
{  printf("\n\n     NAZWISKO           WIEK\n\n");
   while (poc) {
      printf("%20s  %3d\n",poc->nazw, poc->wiek);
      poc=poc->nast;
   }
}
```